Thursday

DATE 02/03/17

PAGE NO.

Inheritance-

(1.) One.cpp

```
#include<iostream.h>
#include<conio.h>
class sum
{
 protected: int a, b;
   public : void getnum()
            {
             cout<<"Enter two values:";
             cin>>a>>b;
            }
             void showtot()
            {
             cout<<a+b;
            }
};
```

(2.) Two.cpp.

```
#include<iostream.h>
#include<conio.h>
#include "one.cpp"
void main()
{
 sum s1, s2;
 s1.getnum();
 s2.getnum();
 s1.showtot();
 s2.showtot();
}
```

Teacher's Signature

e.g.→
```
#include<iostream.h>
#include<conio.h>
class xyz:public sum
{
    public:void multi()
        {
         cout<<a*b;
        }
};
        void main()
        {
         xyz ob;
         ob.getnum();
         ob.showtot();
         ob.multi();
        }
```

Monday
6-3-17

Inheritance-

Inheritance is a process of creating new classes called child, sub or derived from existing class. These existing classes are known as parent, super or base classes.

The derived class inherits all the capabilities of the base class and it can also add new features and refinements of its own, the base class remains unchanged.

Teacher's Signature

It has several advantages to offer most important is that its parents code reusability.

Once a base class is written and debugged it need not to be rewrite it can be adopted to work in different situations.

Syntax:-

```
class childname : parentname
{
  ___
  ___
};
```

e.g.→

```
#include <iostream.h>
#include <conio.h>
class student
{
    protected: int rn;
    char name[20];
    public : void getdata()
            {
             cout<<"Enter name:";
             cin>>name;
             cout<<"Enter roll number:";
             cin>>rn;
            }
};
```

Teacher's Signature

```
class marks: public student
{
    int p,c,m;
    public:void getmarks()
           {
           cout<<"Enter marks:";
           cin>>p>>c>>m;
           }
           void shower()
           {
           cout<<rn<<" "<<name<<" "<<(p+c+m)/3.0;
           }
};
void main()
{
 marks st;
 clrscr();
 st.getdata();
 st.getmarks();
 st.shower();
 getch();
}
```

Teacher's Signature

Tuesday
DATE: 7/3/17
PAGE NO.:

Visibility of Super Class Members in Sub Class:-

Visibility in sub class / Inherited by	Private	Protected	Public
Privately	Private	Private	Private
Protectly	Private	Protected	Protected
Publicly	Private	Protected	Protected

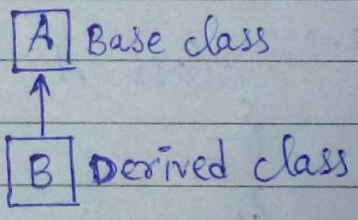
Types of Inheritance:

(1.) Simple and Single Inheritance-

Derivation of a class from only one base class is known as single inheritance.

e.g.→

```
class A
{
  ---
};
class B:A
{
  ---
};
```



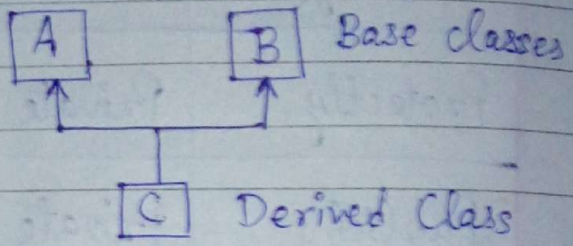
Teacher's Signature

(2.) Multiple Inheritance-

Derivation of a class from more than one base class is known as multiple inheritance.

eg.→

```
class A
{
};
class B
{
};
class C: A, B
{
};
```

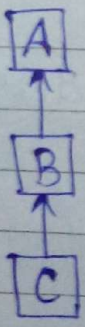


(3.) Multi-level Inheritance-

Derivation of a class from other derived class is known as multi-level inheritance.

eg.→

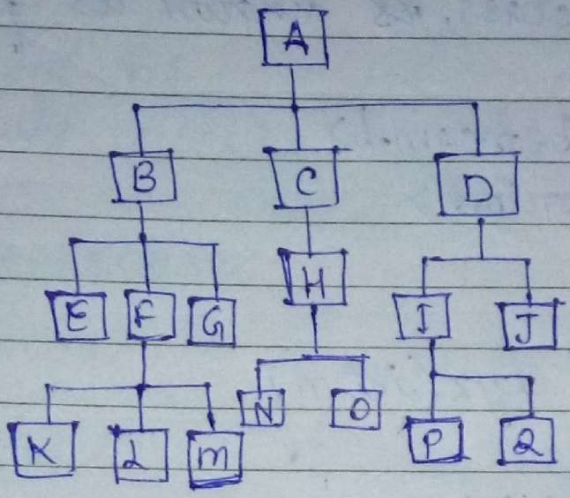
```
class A.
{
};
class B:A
{
};
class C:B
{
};
```



Teacher's Signature

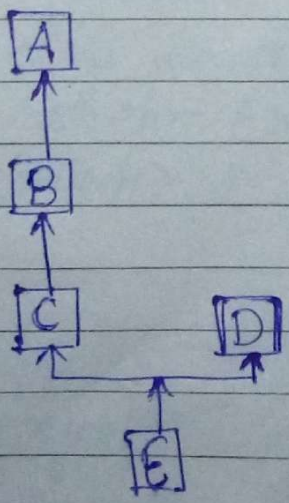
(4.) Hierarchical Inheritance–

Derivation of several classes from a single base class. It represent tree structure This type of inheritance is hierarchical.



(5.) Hybrid Inheritance–

This type of inheritance is a combination of other types of inheritance. For instance a single subclass may inherit properties from multiple base classes which in turn is inheriting from single base class.



Teacher's Signature

Wednesday

DATE 8/3/17

PAGE NO.:

Function Overriding -

If we define a function with same name and same signature in a sub or child or derived class that is already defined in its superclass, is known as function overriding.

e.g.→

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:void xyz(int n)
        {
         cout<<n;
        }
};
class B: public A
{
 public:void xyz(int n)
        {
         cout<<n*n;
        }
};
      void main()
      {clrscr();
       B ob;
       o.xyz(8);
       getch();
      }
```

Class and Pointer-

e.g.→
```
#include<iostream.h>
#include<conio.h>
class A
{
public: int n;
        void xyz()
        {
        cout<<n*n;
        }
};
void main()
 {
  A ob, *k;
  k=&ob;
  ob.n=10;   || *k.n=10;   || k->n=10;
  ob.xyz();  || *k.xyz();  || k->xyz();
  getch();
 }
```

*→ We can pass address of child class object into the parent class pointer.

e.g.→
```
#include<iostream.h>
#include<conio.h>
class A
{
public: int n;
        void xyz()
```

```
        {
        cout<<n*n;
        }
};
class B: public A
{
  public: int m;
          void pk()
          {
          cout<<m+n;
          }
};
   void main()
   {clrscr();
✓ A *k;
✓ B ob;
✓ k=&ob;
x k->m=26;
x k->pk();
  getch();
}
```

```
✓ B *k;
✓ B ob;
✓ k=&ob;
✓ k->m=26;
✓ k->pk();
  getch();
```

Teacher's Signature

Thursday

DATE 9/3/17

PAGE NO.

① Early Binding/Static Binding/Compile time Binding-
Most of the function calls the compiler encounters a direct function call is a statement that directly calls a function.

e.g.→

```
#include<iostream.h>
#include<conio.h>
class A
{
    public: void xyz()
            {
              cout<<"Member of base class";
            }
};
class B: public A
{
        public: void xyz()
                {
                  cout<<"Member of derived class";
                }
};
void main()
{
  A *k;
  B ob;
  clrscr();
  k=&ob;
  k->xyz();
  getch();
}
```

Output-

Member of base class

Teacher's Signature

`k->xyz();` explanation-

In this statement when the compiler compile this program, compiler replaces this function call by xyz() function of class A, because *k pointer is type of A. This type of function binding is known as static/early or compile time binding.

(2.) Late/Dynamic or Run time Binding-

To avoid static or early binding the concept of virtual function is used in C++ that stops the early binding and forces for late/dynamic or runtime binding.

e.g.-

```
#include<iostream.h>
#include<conio.h>
class A
{
    public: virtual void xyz()
        {
        cout<<"Member of base class";
        }
};
class B: public A
{
    public: void xyz()
        {
        cout<<"Member of derived class";
        }
```

Teacher's Signature

```
};
    void main()
    {
      A *k;
      B ob;
      clrscr();
      k=&ob;
      k->xyz();
      getch();
    }
```

Output
Member of base class

`k->xyz();` explanation-

In late binding, function call is resolved at runtime compiler determines the type of object at runtime and then binds the function call.

Virtual Function-

Virtual function is a function in base class which is overrided in derived class and which tells the compiler to perform late binding on this function.

Pure Virtual Function-